УДК: УДК 794.9:004.94
DOI: 10.66523/gici.2026.v1.3

НАУЧНАЯ РЕДАКЦИИ OPEN ACCESS

Визуализация облаков точек в Unity с использованием Gaussian Splatting

Игнатенко И. Д.,[1] Оганесян П. А.[2]

[1] Игнатенко — ЮФУ, Ростов-на-Дону, Россия МГУ-ППИ, Шэнчжэнь, Китай, ЮФУ, Ростов-на-Дону, Россия

Автор для переписки: Оганесян П. А. | oganesian@gamedev.science

Поступила в редакцию: 28.10.2025 | **Принята к публикации:** 08.02.2026 | **Опубликована:** 28.04.2026

Ключевые слова: Gaussian Splatting, Unity, облака точек, визуализация.

Финансирование: *исследование не имело спонсорской поддержки.*

Конфликт интересов: *авторы заявляют об отсутствии конфликта интересов.*

Аннотация

В статье рассматривается современный подход к визуализации облаков точек с помощью метода 3D Gaussian Splatting в игровом движке Unity. Описан принцип построения гауссиан на основе разреженного облака точек, формируемого после применения Structure from Motion, приводится математическая модель трехмерного гауссиана как базового примитива для визуализации. Рассматривается процесс реализации плагина для Unity, обеспечивающего эффективный рендеринг облаков точек на основе метода ГС. Реализованный плагин задействует GPU эффективнее референсной реализации за счет применения сортировки структур, хранящих информацию о гауссианах, в памяти GPU. При реализации учтены особенности Direct3D 11 и Direct3D 12. Приводятся результаты визуализации и сравнительного анализа производительности для предложенной и референсной реализаций.

Point Cloud Visualization in Unity Using Gaussian Splatting

Ignatenko I.D., Oganesian P.A. Corresponding author: Oganesian P.A. | oganesian@gamedev.science

This paper considers a modern approach to point cloud visualization using the 3D Gaussian Splatting method in the Unity game engine. The principle of constructing Gaussians from a sparse point cloud generated via Structure from Motion is described, and a mathematical model of the three-dimensional Gaussian as the basic primitive for visualization is presented. The paper discusses the implementation of a Unity plugin that provides efficient rendering of point clouds based on the Gaussian Splatting (GS) method. The implemented plugin utilizes the GPU more efficiently than the reference implementation by applying sorting of the structures that store Gaussian data in GPU memory. The implementation takes into account the specific features of Direct3D 11 and Direct3D 12. Visualization results and a comparative performance analysis of the proposed and reference implementations are presented.

Keywords: *Gaussian Splatting, Unity, point clouds, visualization.*

Для цитирования: *Визуализация облаков точек в Unity с использованием Gaussian Splatting. Международное сетевое научное рецензируемое издание "Геймификация, игры, креативные индустрии" 2026. Т. 1, № 1. С. 27-32 https://doi.org/10.1155/10.66523/gici.2026.v1.3 / For citation: Ignatenko I.D., Oganesian P.A. Point Cloud Visualization in Unity Using Gaussian Splatting International Peer-Reviewed Online Scientific Journal "Gamification, Games, Creative Industries", 2026, vol. 1, no. 1, pp. 27-32. https://doi.org/10.66523/gici.2026.v1.3*

Введение

Традиционный механизм рендеринга сцен в трехмерных видеоиграх основан на работе с полигональными моделями, текстурами и шейдерами [1,2]. Данный подход называется пайплайном растеризации и, упрощенно, может быть описан как процесс преобразования данных о геометрии сцены в упорядоченную двумерную матрицу пикселей в экранном пространстве, который пользователь видит на своем мониторе. Каждый пиксель характеризуется координатами X, Y и компонентами цвета RGB. В последние годы популярность набирают методы рендеринга, использующие в качестве исходных данных облака точек вместо полигональных сеток. Такие облака могут быть получены как с помощью специальных сенсоров (лидаров, датчиков глубины), так и алгоритмически, в частности, с помощью фотограмметрии по набору обычных фотоснимков. Этот формат данных широко используется в различных областях — от систем автономного вождения до потребительских устройств, включая смартфоны с несколькими камерами или датчиками глубины. Особенный интерес вызывает метод рендеринга 3D Gaussian Splatting [3] (Гауссовы сплаты, далее для краткости будем использовать аббревиатуру ГС, так как в русском языке пока не появилось устойчивое выражение для данной техники), позволяющий визуализировать облака точек с высоким качеством в режиме, приближенном к реальному времени.

Дополнительный импульс развитию технологии ГС дает распространение генеративных нейросетей [4]. Генерация трехмерного контента в формате облаков точек на данный момент реализуется проще, чем создание полигональных моделей, так как представление сцены в формате набора точек ближе к растровому изображению или тензорному полю данных, с которыми эффективно работают современные архитектуры диффузных нейросетей.

Польза данного подхода для создания видеоигр заключается в простоте подготовки данных, что позволяет пользователю по инструкции сделать несколько фотографий, пригодных для фотограмметрии, и получить визуализацию объекта из реального мира в игровом пространстве, либо сгенерировать объект при помощи ИНС.
В данной работе рассматривается создание плагина для платформы Unity, реализующего эффективный рендеринг облаков точек на основе метода ГС.

Метод построения гауссиан

Входными данными для алгоритма являются набор фотографий статической сцены и данные о камерах, откалиброванных при помощи метода Structure from Motion (SfM) [5]. Побочным эффектом от применения SfM является разреженное облако точек, аппроксимирующее геометрию сцены. На этапе инициализации в этих точках строится набор особых примитивов. Целью дальнейших этапов работы алгоритма подготовки сцены является оптимизация полученного представления сцены для обеспечения возможности синтеза высококачественных рендеров с новых ракурсов. Для достижения этой цели необходимо использовать примитивы, обладающие свойствами дифференцируемых пространственных представлений. При этом представление должно быть достаточно неструктурированным и заданным явно для обеспечения возможности быстрой визуализации. В качестве примитива, удовлетворяющего этим требованиям, используется частный вариант гауссовой функции - трехмерный гауссиан. Функция, задающая гауссиан в n-мерном евклидовом пространстве приведена в формуле 1.

$$G(x) = \exp(-x^T Ax + s^T x) \qquad (1)$$

Здесь $x = (x_1, \dots, x_n)$ - вектор столбец из n компонентов, A - симметричная положительно определенная матрица, задающая высоту «колокола», а - вектор сдвига. Такое представление имеет сходство с подходами на основе точек в двухмерном пространстве, каждая из которых является окружностью с нормалью. Однако, учитывая степень разреженности точек после применения метода Structure from Motion, вычисление нормалей может быть затруднительно. Вместо этого, геометрия целевой сцены аппроксимируется набором трёхмерных гауссиан, не требующим вычисления нормалей.

В работе [6] была предложена функция, приведённая в формуле 2 и задающая гауссиан с центром в точке μ при помощи трёхмерной ковариационной матрицы Σ, определённой в мировом пространстве.

$$G(x-\mu) = e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

Для проекции гауссиан на плоскость применяется формула 3:

$$\Sigma' = JW\Sigma W^T J^T \qquad (3)$$

Здесь W - матрица вида (view matrix), J - якобиан аффинной аппроксимации проекционного преобразования, имеющий следующий вид:

$$J = \begin{bmatrix} focal_x/t_z & 0 & -(focal_x \cdot t_x)/(t_z \cdot t_z) & 0 \\ 0 & focal_y/t_z & -(focal_y \cdot t_y)/(t_z \cdot t_z) & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Здесь $t = pW$ позиция гауссиана в мировом пространстве, а Σ'- позиция гауссиана в пространстве вида. В работе [7] было установлено, что у матрицы можно отбросить 2 и 3 строки и столбцы без потери необходимых для проекции свойств.

Для улучшения результатов аппроксимации геометрии и световых полей сцены необходима оптимизация ковариационной матрицы гауссиан. Очевидным решением является прямая оптимизация матрицы. Однако ковариационная матрица сохраняет необходимые свойства только при условии положительной полуопределённости, а использование соответствующих ограничений совместно с методом градиентного спуска, применяемым при оптимизации, не тривиально и может привести к некорректным результатам во время работы [3].

Поэтому необходимо использовать другое представление параметров гауссиан. Представим матрицу поворота в следующем виде:

$$R = \begin{bmatrix} 1-2(q_j^2+q_k^2) & 2(q_iq_j-q_kq_r) & 2(q_iq_k+q_jq_r) \\ 2(q_iq_j+q_kq_r) & 1-2(q_i^2+q_k^2) & 2(q_jq_k-q_iq_r) \\ 2(q_iq_k-q_jq_r) & 2(q_jq_k+q_kq_r) & 1-2(q_i^2+q^2) \end{bmatrix}$$

где $q = (i, j, k, r)$ - кватернион, описывающий вращение соответствующего гауссиана. А матрицу масштабирования представим в виде

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

где $s = (s_x, s_y, s_z)$ - вектор, задающий масштаб гауссиана в трёхмерном пространстве.

Пусть матрица комбинированного преобразования M имеет вид, представленный в формуле 4.

$$M = RS$$

Тогда по определению ковариационная матрица Σ трёхмерного гауссиана имеет вид (5).

$$\Sigma = MM^T \quad (5)$$

По свойству транспонированных матриц получаем, что транспонированное произведение матриц равно произведению транспонированных матриц, взятых в обратном порядке, как показано в (6):

$$\Sigma = RSS^TR^T \quad (6)$$

Такое представление параметров гауссиан позволяет производить их независимую оптимизацию, что также невозможно при прямой оптимизации ковариационной матрицы Σ трёхмерного гауссиана.

Помимо ковариационной матрицы, при помощи метода стохастического градиентного спуска также производится оптимизация таких параметров гауссиан как вектора координат в трёхмерном пространстве *p*, значения прозрачности *a* и коэффициентов сферических гармоник *SH* - особых функций, использующихся для аппроксимации цвета гауссиан в зависимости от угла обзора. Для ограничения множества возможных значений прозрачности до диапазона и повышения плавности градиентов применяется сигмоидная функция активации. По схожим причинам к вектору масштаба *s* применяется экспоненциальная функция активации. В качестве функции потерь используется средняя абсолютная ошибка $\mathcal{L}_1$ с дополнительным членом - значением метрики D-SSIM [8] и коэффициентом регуляризации λ = 0.2, как показано в формуле (7).

$$\mathcal{L} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{D-SSIM}$$

В ходе процесса оптимизации выполняется адаптивный контроль плотности гауссиан. Эта операция позволяет устранить участки сцены, в которых геометрия аппроксимируется недостаточным или избыточным количеством гауссиан.

Визуализация

Программная реализация была выполнена для платформы Unity на языках C# и HLSL и включает в себя и подготовку данных, описанную в разделе II, и визуализацию. После завершения процесса сортировки гауссиан по возрастанию расстояния до активной камеры начинается следующий этап - вычисление параметров гауссиан, необходимых для визуализации сцены. Выполнение этого этапа процесса начинается с загрузки всех данных гауссиан каждым потоком ядра вычислительного шейдера GSProcess.

На следующем шаге при помощи умножения на матрицу, полученную в результате комбинирования проекционного преобразования и преобразования координат объектов в пространство камеры, вычисляется положение центральной точки гауссиана в мировом пространстве. Возможна также дополнительная модификация параметров гауссиан в соответствии с аффинными преобразованиями, выполненными над родительским объектом на игровой сцене.

Если центральная точка гауссиана после проекции в мировое пространство оказывается позади камеры, то вычисление параметров этого гауссиана прекращается для повышения производительности алгоритма. В противном случае, на следующем этапе на основе значений вектора масштабирования и кватерниона поворота по формуле (4) строится матрица RS, задающая комбинированное преобразование вращения и масштаба гауссиана в трёхмерном пространстве. К ней также применяется преобразование, переводящие координаты в мировые. После этого по формуле (5) вычисляются элементы ковариационной матрицы Σ гауссиана.

Матрица ковариации симметрична, что позволяет хранить не все её элементы, а только верхнюю треугольную часть. В оригинальном алгоритме для хранения матрицы использовался динамический массив типа «float*», который для экономии регистров был заменен на два вектора типа «float3» при переносе реализации на язык HLSL.
Затем, по формуле 3 выполняется проекция матрицы Σ гауссиана в двухмерное пространство камеры. К полученной матрице Σ' дополнительно применяется низкочастотный пропускной фильтр

$$0.3 \cdot I$$

где I – диагональная матрица размера 2×2.
Применение фильтра обусловлено необходимостью обеспечения обратимости матрицы Σ' в дальнейших шагах алгоритма визуализации.

После этого, у матрицы Σ', имеющей структуру, приведённую в формуле (9), по формулам (10) и (11) вычисляются собственные значения λ_1 и λ_2 соответственно.

$$\Sigma' = \begin{bmatrix} a & b \\ b & c \end{bmatrix} \quad (9)$$

$$\lambda_1 = \frac{a + c + \sqrt{a^2 - 2ac + c^2}}{2} \quad (10)$$

$$\lambda_2 = \frac{a + c - \sqrt{a^2 - 2ac + c^2}}{2} \quad (11)$$

Наибольшее из найденных собственных значений используется для построения конфигурации эллипса. По формуле (12) строится матричное представление конического сечения, которое мы далее будем называть конической матрицей, в согласии с в ряде работ нотацией.

$$conic = (\Sigma')^{-1} = \frac{1}{ac - b^2}\begin{bmatrix} c & -b \\ -b & a \end{bmatrix}$$

Применение низкочастотного фильтра гарантирует, что матрица Σ' невырождена, то есть её определитель не равен нулю. На следующем этапе при помощи оценки значения вещественных сферических гармоник происходит вычисление цвета гауссиан в зависимости от направления взгляда камеры.

Наконец, все вычисленные данные гауссиана помещаются в отдельный буфер данных, состоящий из более компактных структур по сравнению с исходным представлением гауссиан, чтобы сократить объем данных, пересылаемых внутри графического конвейера.

После завершения вычисления параметров гауссиан можно приступать к отрисовке подготовленных буферов данных. Выполнение этого этапа отри-совки реализовано при помощи двух шейдерных подпрограмм «RenderSplats» и «Composite». Вершинный шейдер первой программы выполняет следующий набор операций:

- извлекает данные конкретного гауссиана из буфера в отсортированном порядке при помощи переиндексации через ранее сохраненные значения;
- передаёт во фрагментный шейдер вычисленное значение цвета гауссиана и его коническую матрицу;
- вычисляет положение центра гауссиана в экранных координатах;
- строит экранный четырёхугольник вокруг центра гауссиана, масштабируя его в соответствии с вычисленным значением радиуса;

После завершения процесса построения набора четырёхугольников начинается следующий этап работы шейдерной программы. Пиксельный шейдер вычисляет расстояние d от центра эллипса до текущего фрагмента в экранных координатах. Плотность эллиптического гауссиана с центром в точке p рассчитывается при помощи формулы 13, представленной в работе [9].

$$G_V(x - p) = \frac{1}{2\pi|V|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - p)^T V^{-1}(x - p)\right) \quad (13)$$

Здесь *V* – матрица ковариации соответствующего гауссиана. Реализация этой функции в данной работе отличается от приведённой в формуле (13) из работы [9]. В алгоритме оптимизации представления сцены методом ГС не используется коэффициент перед экспонентой. Для обеспечения корректности визуализации сцены вклад этого коэффициента не учитывается и в шейдере. В качестве значения *p* используется (0,0). Упрощено выражение в показателе степени экспоненты. Таким образом, если коническая матрица V^{-1} гауссиана имеет вид, представленный в формуле (14), то итоговое затухание плотности гауссиана в его произвольной точке x оценивается при помощи функции, приведённой в формуле (15).

$$V^{-1} = \begin{bmatrix} a & b \\ b & c \end{bmatrix} \quad (14)$$

$$G_V(x) = \exp\left(-\frac{1}{2}(ax_0^2 + 2bx_0y_0 + cy_0^2)\right) \quad (15)$$

Здесь $x = (x_0, y_0)$. Фрагмент может быть отброшен, если в результате проведённых вычислений его значение прозрачности не превышает $\frac{1}{255}$

Вычисление комбинированного цвета пикселя *C* путем смешивания отсортированного набора из *N* перекрывающих его гауссиан в методе ГС осуществляется по формуле (16)

$$C = \sum_{i \in N} c_i \alpha_i \prod_{j=1}^{i-1}(1 - \alpha_j) \quad (16)$$

Здесь c_i – значение цвета i-го гауссиана в формате RGB, а α_i – результат комбинирования накопленного значения непрозрачности пикселя во время отрисовки и значения прозрачности гауссиана, рассчитанного во время оптимизации представления сцены. Во фрагментном шейдере одна из частей этой формулы реализована при помощи умножения цвета гауссиана на его значение прозрачности. Другая часть формулы реализована при помощи функции смешивания, выполняющей накопление значений прозрачности результатов визуализации других гауссиан.

Результаты визуализации различных объектов представлены на рисунках 1 и 2.

Рисунок 1. Пример визуализации отдельного объекта, полученного из облака точек.

Рисунок 1. Пример визуализации отдельного объекта, полученного из облака точек.

Для оценки применимости разработанного алгоритма для приложений, работающих в режиме, близком к реальному времени, были проведены замеры производительности на различных конфигурациях оборудования. Данная реализация алгоритма в первую очередь зависит от производительности GPU, поэтому в таблице 1 приводятся результаты запуска визуализации для одного и того же набора входных данных на разных видеоускорителях. Рассматривались как потребительские GPU серии RTX, так и профессиональные устройства серии Quadro. Оценивалось количество кадров в секунду (FPS), получаемое при запуске сцены в редакторе Unity. Также проводилось сравнение с референсной реализацией [3]. Данные представлены в формате , сначала указан FPS, достигнутый в данной реализации, а затем FPS, полученный в референсной реализации. Для каждой тестовой сцены указано количество используемых гауссиан. Исходный код проекта и тестовые сцены доступны в репозитории [10]. Как видно, предложенный алгоритм показывает результаты, близкие к референсным, однако для более мощных видеокарт оказывается эффективнее. При этом значительной разницы в качестве полученной картинки замечено не было. Прирост производительности был достигнут, в частности, за счет реализации алгоритмов сортировки структур данных гауссиан на GPU. Для версии, работающей на Direct3D 11, была применена битонная сортировка, а для версии, использующей Direct3D 12, применялась поразрядная сортировка на основе побитового представление ключей сортируемых данных.

Сцена/GPU	NVIDIA Quadro GV100	NVIDIA RTX 3070M	NVIDIA RTX 2060 Super
toy, 7k	261/63	139/144	160/242
stump, 7k	85/62	69/84	70/77
stump, 30k	70/60	53/67	53/60
garden, 7k	80/41	57/67	56/60
garden, 30k	58/45	43/53	44/47
bicycle, 7k	72/31	60/69	44/61
bicycle, 30k	47/31	37/42	29/39

Таблица 1. Сравнение FPS для различных GPU.

Заключение

Результаты данной работы могут применяться для визуализации сцен и объектов методом ГС для игр и приложений на Unity. Следует отметить ряд ограничений, возникающих при такой визуализации. В частности, встроенная физика игрового движка не может напрямую взаимодействовать с гауссианами, как и навигационные сетки, и система анимации. Для преодоления таких ограничений можно применять прокси-коллайдеры, например, выпуклые оболочки, и другие объекты, переносящие часть свойств гауссиан в привычные движку объекты. Интересным решением видится комбинация ГС для фотореалистичных окружений и полигональных сеток для интерактивных игровых объектов.

Список литературы

[1] T. Akenine-Möller, E. Haines, N. Hoffman, Real-Time Rendering, 4th ed. Boca Raton, FL: CRC Press, 2018.
[2] J. F. Hughes et al., Computer Graphics: Principles and Practice, 3rd ed. Boston, MA: Addison-Wesley, 2013
[3] B. Kerbl, G. Kopanas, T. Leimkuehler, and G. Drettakis.. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. ACM Trans. Graph. 42, 4, Article 139 (August 2023), 14 pages. https://doi.org/10.1145/3592433
[4] B. Poole et al., DreamFusion: Text-to-3D using 2D Diffusion. Proc. Int. Conf. Learn. Represent. (ICLR), 2023.
[5] A. Eltner and G. Sofia, Structure from motion photogrammetric technique, in Developments in Earth Surface Processes, vol. 23, P. Tarolli and S. M. Mudd, Eds. Amsterdam, Netherlands: Elsevier, 2020, pp. 1-24.
[6] M. Zwicker, H. Pfister, J. van Baar, and M. Gross, Surface splatting, in Proc. 28th Annu. Conf. Comput. Graph. Interact. Tech. (SIGGRAPH '01). New York, NY, USA: ACM, 2001, pp. 371-378, doi: 10.1145/383259.383300.
[7] M. Zwicker, H. Pfister, J. van Baar, and M. Gross, EWA volume splatting, in Proc. IEEE Vis. (VIS '01). Washington, DC, USA: IEEE Computer Society, 2001, pp. 29-36.
[8] A. H. Baker, A. Pinard and D. M. Hammerling, On a Structural Similarity Index Approach for Floating-Point Data, in IEEE Transactions on Visualization and Computer Graphics, vol. 30, no. 9, pp. 6261-6274, Sept. 2024, doi: 10.1109/TVCG.2023.3332843.
[9] V. Yugay, Y. Li, T. Gevers, and M. R. Oswald, Gaussian-SLAM: Photo-realistic Dense SLAM with Gaussian Splatting, arXiv:2312.10070 [cs.CV], Dec. 2023. [Online]. Available: https://arxiv.org/abs/2312.10070
[10] Электронный репозиторий с исходным кодом проекта: https://github.com/COOLIRON2311/Mnemonic